Analysis of Algorithms: Solutions 3

Problem 1
Determine asymptotic upper and lower bounds for each of the following recurrences.

(a) \( T(n) = 27T(n/3) + n \)

\[
\begin{align*}
T(n) & = n + 27T\left(\frac{n}{3}\right) \\
& = n + 27\left(\frac{n}{3} + 27T\left(\frac{n}{3^2}\right)\right) \\
& = n + 27^2T\left(\frac{n}{3^2}\right) \\
& = n + 27\left(\frac{n}{3} + 27^2\left(\frac{n}{3^2} + 27T\left(\frac{n}{3^3}\right)\right)\right) \\
& = n + 27\left(\frac{n}{3} + 27^2\left(\frac{n}{3^2} + 27^3T\left(\frac{n}{3^3}\right)\right)\right) \\
& \quad \ldots \\
& = n + 27^\log_3n \frac{n}{3^{\log_3n}} \\
& = n + 9n + 9^2n + 9^3n + 9^4n + \ldots + 9^{\log_3n}n \\
& = n(1 + 9 + 9^2 + 9^3 + 9^4 + \ldots + 9^{\log_3n}) \\
& = \frac{9^{\log_3n+1} - 1}{9 - 1} \\
& = \frac{9n^2 - 1}{8} \\
& = \Theta(n^3)
\end{align*}
\]
(b) \( T(n) = 2T(n/3) + n^3 \)

\[
T(n) = n^3 + 27\left(\frac{n}{3}\right)^3 + 27T\left(\frac{n}{3^2}\right)
\]

\[
= n^3 + 27\left(\frac{n}{3}\right)^3 + 2\cdot27^2T\left(\frac{n}{3^2}\right)
\]

\[
= n^3 + 27\left(\frac{n}{3}\right)^3 + 2\cdot27^2\left(\frac{n}{3^2}\right)^3 + 2\cdot27^3T\left(\frac{n}{3^3}\right)
\]

\[
\ldots
\]

\[
= n^3 + 27\left(\frac{n}{3}\right)^3 + 2\cdot27^2\left(\frac{n}{3^2}\right)^3 + 2\cdot27^3\left(\frac{n}{3^3}\right)^3 + \ldots + 2\cdot27^{\log_3 n}\left(\frac{n}{3^{\log_3 n}}\right)^3
\]

\[
= n^3 + n^3 + n^3 + n^3 + \ldots + n^3
\]

\[
= n^3 (\log_3 n + 1)
\]

\[
= \Theta(n^3 \cdot \log n)
\]

(c) \( T(n) = 3T(n/9) + \sqrt{n} \)

\[
T(n) = \sqrt{n} + 3T\left(\frac{n}{9}\right)
\]

\[
= \sqrt{n} + 3\left(\sqrt{\frac{n}{9}} + 3T\left(\frac{n}{9^2}\right)\right)
\]

\[
= \sqrt{n} + 3\sqrt{\frac{n}{9}} + 3^2T\left(\frac{n}{9^2}\right)
\]

\[
= \sqrt{n} + 3\sqrt{\frac{n}{9^2}} + 3^2\left(\sqrt{\frac{n}{9^2}} + 3T\left(\frac{n}{9^3}\right)\right)
\]

\[
= \sqrt{n} + 3\sqrt{\frac{n}{9^2}} + 3^2\sqrt{\frac{n}{9^3}} + 3^3T\left(\frac{n}{9^3}\right)
\]

\[
\ldots
\]

\[
= \sqrt{n} + 3\sqrt{\frac{n}{9^2}} + 3^2\sqrt{\frac{n}{9^3}} + 3^3\sqrt{\frac{n}{9^4}} + \ldots + 3^{\log_3 n}\sqrt{\frac{n}{9^{\log_3 n}}}
\]

\[
= \sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} + \ldots + \sqrt{n}
\]

\[
= \sqrt{n} (\log_3 n + 1)
\]

\[
= \Theta(\sqrt{n} \cdot \log n)
\]
(d) $T(n) = T(\sqrt{n}) + 1$

We “unwind” the recurrence until reaching some constant value of $n$, say, until $n \leq 2$:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 2 \\
T(\sqrt{n}) + 1 & \text{if } n > 2 
\end{cases}$$

For convenience, assume that $n = 2^{2k}$, for some natural value $k$.

$$T(2^{2k}) = 1 + T(\sqrt{2^{2k}})$$
$$= 1 + T(2^{2k-1})$$
$$= 1 + 1 + T(\sqrt{2^{2k-1}})$$
$$= 1 + 1 + T(2^{2k-2})$$
$$= 1 + 1 + 1 + T(\sqrt{2^{2k-2}})$$
$$= 1 + 1 + 1 + T(2^{2k-3})$$
$$\quad \vdots$$
$$= \underbrace{1+1+\ldots+1}_{k} + T(2)$$
$$= k + \Theta(1)$$
$$= \Theta(k)$$

Finally, we note that $k = \lg \lg n$, which means that $T(n) = \Theta(\lg \lg n)$.

(e) $T(n) = T(n - 1) + n^2$

$$T(n) = n^2 + T(n - 1)$$
$$= \underbrace{n^2 + \ldots + n^2}_{n \text{ times}} + T(n - 2)$$
$$= n^2 + (n - 1)^2 + T(n - 2)$$
$$= n^2 + (n - 1)^2 + ((n - 2)^2 + T(n - 3))$$
$$= n^2 + (n - 1)^2 + (n - 2)^2 + T(n - 3)$$
$$\quad \vdots$$
$$= n^2 + (n - 1)^2 + (n - 2)^2 + (n - 3)^2 + (n - 4)^2 + \ldots + 1^2$$
$$= \frac{n(n + 1)(2n + 1)}{6}$$
$$= \Theta(n^3)$$
Problem 2

Consider the following sorting algorithm:

\textbf{STOOGE-SORT}(A, i, j)
2. \textbf{then} exchange $A[i] \leftrightarrow A[j]$
3. \textbf{if} $i + 1 \geq j$
4. \textbf{then} return
5. $k \leftarrow \lfloor (j - i + 1)/3 \rfloor$
6. \textbf{STOOGE-SORT}(A, $i, j - k$) \hspace{1em} \triangleright \text{first two-thirds}
7. \textbf{STOOGE-SORT}(A, $i + k, j$) \hspace{1em} \triangleright \text{last two-thirds}
8. \textbf{STOOGE-SORT}(A, $i, j - k$) \hspace{1em} \triangleright \text{first two-thirds again}

(a) Argue that \textbf{STOOGE-SORT}(A, 1, n) correctly sorts the input array $A[1..n]$.

We prove the correctness of the algorithm by induction. Clearly, the algorithm works for one-element and two-element arrays, which provides the induction base. Now suppose that it works for all arrays shorter than $A[i..j]$ and let us show that it also works for $A[i..j]$.

After the execution of Line 6, $A[i..(j - k)]$ is sorted, which means that every element of $A[(i + k)..(j - k)]$ is no smaller than every element of $A[i..(i + k - 1)]$; we write it as $A[(i + k)..(j - k)] \geq A[i..(i + k - 1)]$. Thus, $A[(i + k)..j]$ has at least $\text{length}(A[(i + k)..(j - k)]) = j - i - 2k + 1$ elements each of which is no smaller than each element of $A[i..(i + k - 1)]$.

After the execution of Line 7, $A[(i + k)..j]$ is sorted, which implies that

1. $A[(j - k + 1)..j]$ is sorted, and
2. $A[(j - k + 1)..j] \geq A[(i + k)..(j - k)]$.

Since $A[(i + k)..j]$ has at least $(j - i - 2k + 1)$ elements no smaller than each element of $A[i..(i + k + 1)]$ and $\text{length}(A[(j - k + 1)..j]) \leq j - i - 2k + 1$, we conclude that

3. $A[(j - k + 1)..j] \geq A[i..(i + k - 1)]$.

Putting together (2) and (3), we conclude that

4. $A[(j - k + 1)..j] \geq A[i..(j - k)]$.

After the execution of Line 8, the array $A[i..(j - k)]$ is sorted. Putting this observation together with (1) and (4), we see that the whole array $A[i..j]$ is sorted.
(b) Give the recurrence for the worst-case running time of STOOGE-SORT and a tight asymptotic (Θ-notation) bound on the worst-case running time.

The algorithm first performs a constant-time computation (Lines 1–5), and then recursively calls itself three times (Lines 6–8), each time on an array whose size is \( \frac{2}{3} \) of the original array’s size. Thus, the recurrence is as follows:

\[
T(n) = 3T\left(\frac{2}{3}n\right) + \Theta(1).
\]

This recurrence describes both the worst-case and best-case running time, since the algorithm’s behavior does not depend on the order of elements in the input array. We use the iteration method to solve it:

\[
\begin{align*}
T(n) &= 1 + 3T\left(\frac{2}{3}n\right) \\
&= 1 + 3 + 9T\left(\frac{4}{9}n\right) \\
&\quad \cdots \\
&= 1 + 3 + 3^2 + \ldots + 3^{\log_3/2}n \\
&\quad \quad = \frac{3^{\log_3/2}n+1 - 1}{3 - 1} \\
&= \Theta(3^{\log_3/2}n) \\
&= \Theta(3^{(\log_3 n)/(\log_3 3/2)}) \\
&= \Theta(n^{1/(\log_3 3/2)}) \\
&= \Theta(n^{2.71}).
\end{align*}
\]

(c) Compare the worst-case running time of STOOGE-SORT with that of INSERTION-SORT and MERGE-SORT. Is it a good algorithm?

STOOGE-SORT is slower than the other sorting algorithms. Even INSERTION-SORT has the complexity \( O(n^2) \), which is much better than \( \Theta(n^{2.71}) \).
Problem 3
The following algorithm inputs a natural number $n$ and returns a natural number $m$.

**SLOW-COUNTER($n$)**

for $i \leftarrow 1$ to $n$
    do for $j \leftarrow 1$ to $n$
        do $S \leftarrow \emptyset$ ▷ make the set $S$ empty
            for $k \leftarrow 1$ to $i - 1$
                do $S \leftarrow S \cup \{A[k, j]\}$ ▷ add the $A[k, j]$ value to $S$
            for $k \leftarrow 1$ to $j - 1$
                do $S \leftarrow S \cup \{A[i, k]\}$ ▷ add the $A[i, k]$ value to $S$
            $A[i, j] \leftarrow \text{MAX}(S) + 1$
    $m \leftarrow A[n, n]$

return $m$

Give a much faster algorithm that computes the same value $m$.

Every element $A[i, j]$ of the resulting matrix is 1 greater than its preceding neighbors $A[i-1, j]$ and $A[i, j-1]$. For example, if $n = 8$, then the matrix is as follows:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{array}
\]

Thus, $m$ is always $2n - 1$, and we may replace **SLOW-COUNTER** with the following algorithm:

**FAST-COUNTER($n$)**

return $2n - 1$